IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application No.:	10/820,661	Examiner:	Kim, Paul
Atty. Docket No.:	SUN04-0234	Art Unit:	2169
Filed:	07 April 2004	Conf. No.:	8024
Appellant:	Paul A. Martin		

For: *Method and Apparatus for Performing Lock-Free Updates in a Linked List*

APPEAL BRIEF

Sir:

In response to the Notice of Panel Decision from Appeal Brief Review mailed 09 January 2009, and subsequent to the Notice of Appeal filed 03 December 2008, Appellant submits this Appeal Brief to appeal the rejection of claims 1, 3-15, 17-29, and 31-42 under 35 U.S.C. § 103(a) in a Final Office Action mailed 03 September 2008. This Appeal Brief demonstrates that such rejections cannot be sustained because the gap between the prior art and the claimed invention is so great as to render the claims nonobvious to one reasonably skilled in the art.

TABLE OF CONTENTS

THE REAL PARTY IN INTEREST

The real party in interest in this appeal is Sun Microsystems, Inc., the assignee of this application.

RELATED APPEALS AND INTERFERENCES

Appellant is not aware of any appeals or interferences that will affect directly, will be affected directly by, or will otherwise have bearing on the decision in this appeal.

For completeness, the Appellant notes the following applications which are related to this application:

1. Pending U.S. app. no. 11/125,910, filed on 10 May 2005 (Atty. Docket No.: SUN04-0263-US-CIP);

STATUS OF CLAIMS

The status of the claims is as follows:

Claims pending:	1, 3-15, 17-29, and 31-42
Claims rejected:	1, 3-15, 17-29, and 31-42
Claims objected to:	None.
Claims cancelled:	2, 16, and 30
Claims appealed:	1, 3-15, 17-29, and 31-42

STATUS OF AMENDMENTS

All amendments have been entered. A copy of the rejected claims is attached as appendix A.

SUMMARY OF THE CLAIMED SUBJECT MATTER

The claims in the instant application are directed toward a method and an apparatus for performing lock-free updates to nodes in a linked list.

As described in the Related Art section of the instant application, "lock-free" linked lists have been developed which can be used more efficiently in a multi-threaded environment than those linked list implementations using locks. However, while the nodes in the existing lock-free linked lists can be inserted, searched for, or deleted, the values in nodes in the existing lock-free linked lists cannot be *changed or updated* in a lock-free manner. This makes such lock-free linked lists (as well as more complex data structures formed from them) less useful in applications where nodes are updated.

The claimed invention addresses this problem by performing lock-free updates of existing nodes in a linked list. To perform the lock-free update, the system first creates a new node to be added to the linked list, wherein other processes do not possess references to the new node and therefore cannot initially access the new node. Next, the system copies a snapshot of the existing node to the new node, and then updates one or more fields in the new node that correspond to the one or more fields in the existing node. Next, in a single atomic operation the system modifies a next pointer of the existing node to point to the new node and also marks the next pointer to indicate that the existing node is deleted. In this way, the new node becomes part of the linked list and the existing node is deleted in a single atomic operation.

The process of updating fields in a node in a linked list is described in paragraphs [0029]-[0037] and FIGs. 2B-2D and 3 of the instant application; operation of copying a snapshot of an existing node to a new node is described in paragraphs [0038]-[0039] and FIG. 4 of the instant application.

<u>Independent Claim 1: A Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List</u>

The claimed method ("the method") enables lock-free updates to nodes in a linked list. The method is described in paragraphs [0029]-[0039] of the instant application. A system for performing lock-free updates to nodes in a linked list is described in paragraphs [0025]-[0027] of the instant application.

The method involves receiving a reference to an existing node in the linked list, wherein the existing node contains one or more fields to be updated. The method also involves obtaining a new node to be added to the linked list. A linked list node is described in paragraph [0028] of the instant application and a linked list is described in paragraph [0029] of the instant application.

The method further involves: (1) copying a snapshot of the existing node into the new node, which involves examining the next pointer of the existing node to determine if the existing node has been deleted; (2) updating one or more fields in the new node that correspond to the one or more fields in the existing node to be updated; (3) performing a single atomic operation that modifies the next pointer of the existing node to point to the new node and also marks the next pointer to indicate that the existing node is deleted; and (4) splicing the existing node out of the linked list by atomically modifying the next pointer of a node immediately preceding the existing node in the linked list to point to the new node, instead of pointing to the existing node. The copying operation is described in paragraphs [0038]-[0039] of the instant application. The updating operation is described in paragraph [0032] of the instant application. The single atomic operation that both modifies and marks the next pointer of the existing node is described in paragraphs [0033]-[0034] of the instant application. The splicing operation is described in paragraph [0036] of the instant application.

Dependent Claim 3: A Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List

Dependent claim 3 depends upon independent claim 1. In the claimed method, if a process that deleted the existing node does not perform the splicing operation, another process, which subsequently detects that the existing node has been deleted, performs the splicing operation. Using another process to perform the splicing operation is described in paragraph [0037] of the instant application.

Dependent Claim 4: A Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List

Dependent claim 4 depends upon independent claim 1. During the copying operation, the claimed method further involves copying the contents of the existing node to the new node and examining the next pointer of the existing node to determine if the existing node has been deleted. If so, the claimed method takes a remedial action. Otherwise, no remedial action is taken. This more detailed copying operation involving a possible remedial action is described in paragraphs [0038]-[0039] of the instant application.

Dependent Claim 5: A Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List

Dependent claim 5 depends upon dependent claim 4. In the claimed method, taking the remedial action involves following the next pointer of the existing node in an attempt to find an updated version of the existing node. If an updated version of the existing node is found, the claimed method further involves copying a snapshot of the updated version of the existing node to the new node. If an updated version of the existing node is not found, the claimed method further involves indicating that the remedial action fails. This remedial action is described in paragraph [0039] of the instant application.

<u>Dependent Claim 6: A Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List</u>

Dependent claim 6 depends upon independent claim 1. The claimed method further involves deleting a target node from the linked list by: (1) receiving a reference to the target node to be deleted from the linked list; (2) atomically marking a next pointer in the target node to indicate that the target node is deleted; and (3) atomically modifying the next pointer of a node immediately preceding the target node in the linked list to point to a node immediately following the target node in the linked list, instead of pointing to the target node, thereby splicing the target node out of the linked list. This deletion operation is described in paragraphs [0041] and [0043] of the instant application.

<u>Dependent Claim 7: A Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List</u>

Dependent claim 7 depends upon dependent claim 6. In the claimed method, after the target node is spliced out of the linked list, taking the remedial action involves modifying the next pointer of the target node so that the next pointer remains marked but points to a node immediately preceding the target node instead of the node immediately following node the target node in the linked list. This remedial action is described in paragraph [0045] of the instant application.

<u>Dependent Claim 8: A Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List</u>

Dependent claim 8 depends upon independent claim 1. The claimed method further involves inserting an additional node into the linked list by: (1) identifying a node immediately preceding the additional node in the linked list;

(2) identifying a node immediately following the additional node in the linked list; and (3) splicing the additional node into the linked list by: setting the next pointer for the additional node to point to the immediately following node, and atomically updating the next pointer of the immediately preceding node to point to the additional node. This insertion operation is described in paragraphs [0046]-[0047] of the instant application.

<u>Dependent Claim 9: A Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List</u>

Dependent claim 9 depends upon independent claim 1. The claimed method further involves reading a snapshot of multiple fields from a target node in the linked list by: (1) reading the multiple fields from the target node; (2) examining the next pointer of the target node to determine if the target node has been deleted; and (3) if so, taking a remedial action, otherwise, not taking the remedial action. The operation of reading the snapshot of multiple fields is described in paragraphs [0051]-[0052] of the instant application.

<u>Dependent Claim 10 A Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List</u>

Dependent claim 10 depends upon dependent claim 9. In the claimed method, taking the remedial action involves following the next pointer of the target node in an attempt to find an updated version of the target node. If an updated version of the target node is found, the claimed method repeats the process of reading a snapshot of the multiple fields from the updated version of the target node. If an updated version of the existing node is not found, the clamed method indicates that the remedial action fails. This remedial action is described in paragraph [0052] of the instant application.

Dependent Claim 11 A Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List

Dependent claim 11 depends upon independent claim 1. In the claimed method, atomically modifying the next pointer of the existing node to indicate that the existing node is deleted involves setting a "deleted bit" in the next pointer. This deleted bit is described in paragraph [0034] of the instant application.

Dependent Claim 12 A Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List

Dependent claim 12 depends upon independent claim 1. In the claimed method, while atomically modifying the next pointer of the existing node, if the next pointer indicates that the existing node is already deleted, the atomic modification operation fails and the claimed method further involves taking a remedial action to deal with the fact that the existing node is already deleted. The process of detecting the atomic modification operation failure and taking remedial action is described in paragraph [0035] of the instant application.

Dependent Claim 13: A Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List

Dependent claim 13 depends upon independent claim 1. In the claimed method, a given node in the linked list includes: (1) a key that contains an identifier for the given node; (2) one or more fields containing data values or pointers to data values associated with the given node; and (3) a next pointer that contains the address of a node that immediately follows the given node in the linked list, and that also contains a deleted indicator, which indicates whether the given node has been deleted. This node structure in the linked list is described in paragraph [0028] of the instant application.

<u>Dependent Claim 14: A Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List</u>

Dependent claim 14 depends upon independent claim 1. The claimed method further involves periodically performing a garbage-collection operation to reclaim deleted nodes that have become unreachable. The garbage-collection operation is described in paragraph [0012] of the instant application.

<u>Independent Claim 15 and Dependent Claims 16-28: A Computer-Readable Storage Medium Storing Instructions when Executed by a Computer Cause the Computer to Perform a Method for Performing a Lock-Free Update to One or More Fields in an Existing Node in a Linked List</u>

Much of the subject matter in independent claim 1 and dependent claims 2-14 also appears in independent claim 15 and dependent claims 16-28, respectively, as applied to a computer-readable storage medium. Aside from the computer readable storage medium, which is described in paragraph [0024] of the instant application, the remaining subject matter of claims 1-14, as summarized above, is sufficient to establish patentability. Appellant therefore does not repeat the above description.

<u>Independent Claim 29 and Dependent Claims 30-42: An Apparatus that Performs a Lock-Free Update to One or More Fields in an Existing Node in a Linked List</u>

Much of the subject matter in independent claim 1 and dependent claims 2-14 also appears in independent claim 29 and dependent claims 30-42, respectively, as applied to an apparatus. Aside from the apparatus, which is described in paragraphs [0025]-[0027] of the instant application, the remaining

subject matter of claims 1-14, as summarized above, is sufficient to establish patentability. Appellant therefore does not repeat the above description.

GROUNDS OF REJECTION PRESENTED FOR REVIEW

In the Official Action mailed on 03 September 2008 (hereinafter "0903 OA"), Examiner reviewed claims 1, 3-15, 17-29, and 31-42. Examiner rejected claims 1, 4, 6-9, 11, 13-14, 15, 18, 20-23, 25, 27-28, 29, 32, 34-37, 39, and 41-42 under 35 U.S.C. § 103(a) based on McGregor (*Practical C++*, published by Que on 11 August 1999, hereinafter "McGregor") in view of Guthrie (U.S. Patent No. 7,225,210, hereinafter "Guthrie"). Examiner rejected claims 3, 17, and 31 under 35 U.S.C. § 103(a) based on McGregor in view of Guthrie and further in view of Official Notice. Examiner rejected claims 5, 10, 12, 19, 24, 26, 33, 38, and 40 based on McGregor in view of Guthrie and further in view of Lippmann et al. ("C++ Primer," published by Addison Wesley Professional, 2 April 1998, hereinafter "Lippmann").

For the purposes of this appeal, and without admission as to the appropriateness of the other grounds raised by the Examiner, Appellants address the Examiner's reliance on McGregor for disclosing performing an **atomic operation to update a node in a linked list**. More specifically, Appellants will demonstrate that McGregor nowhere discloses an update operation for a linked list that involves performing an atomic operation that **removes** an existing node from the linked list and **adds** a new node to the linked list with updated data in the place of the existing node.

ARGUMENTS

Rejections under 35 U.S.C. § 103(a)

In response to the rejection under 35 U.S.C. § 103(a) in the Final Office Action mailed 03 September 2008 (hereinafter "0903 OA"), Appellant submits that the rejections cannot be sustained because when establishing a prima facie case when rejecting claims under 35 U.S.C. § 103, the Examiner's cited prior art must cover the claimed subject matter.[1] Where the prior art does not cover the claimed subject matter, the Examiner is required to explain the differences:

> The prior art reference (or references when combined) need not teach or suggest all the claim limitations, however, **Office personnel must explain why the difference(s) between the prior art and the claimed invention would have been obvious to one of ordinary skill in the art**; and
>
> The gap between the prior art and the claimed invention **may not be so great as to render the claim nonobvious to one reasonably skilled in the art**.[2]

In the instant case, the gap between the prior art cited by the Examiner and the claimed invention is so great as to render the claims nonobvious to one reasonably skilled in the art.

Overview of McGregor

In the interest of clarifying the arguments against the rejection of the claimed invention using McGregor, we provide an overview of the system disclosed in McGregor. In addition to providing the overview, we briefly identify the limitations of the claimed invention that are missing from the McGregor system.

---

[1] see MPEP § 2141(II)(A)(1)

[2] see MPEP § 2141(III), emphasis added

Generally, McGregor describes a number of standard linked list operations for a doubly linked list. These standard linked list operations include creating a linked list, adding a node to the end of a linked list, inserting a node into a linked list, searching for a node in a linked list, and deleting a node from a linked list.[3] However, McGregor nowhere discloses **updating an existing node** in a linked list. Moreover, McGregor nowhere discloses **the lock-free updating** of an existing node in a linked list. In fact, McGregor nowhere discloses performing any lock-free linked list operations.

Furthermore, when inserting a new node into a linked list, McGregor discloses: (1) allocating a new node; and (2) setting the integer value for the new node.[4] However, McGregor discloses only *initializing* a value in the new node. McGregor nowhere discloses *updating* an existing value in a node in a linked list, particularly where the existing value in the node is copied from an existing node to a newly added node.

Although never describing any sequence of operations for updating a node, McGregor describes performing some of the sub-operations arguably involved in the update operation, such as a "delete" operation and an "insert" operation. However, even for the simple "delete" operation alone, McGregor expressly discloses a **multi-step operation**. For example, consider the following section of McGregor:

> "All you have to do is swap the pointers of the nodes on either side of the node you are deleting. **You swap the pointers in two steps**.
>
> **First**, using extended pointer-to-member notation to access the next node's mypPrev member, assign to it the address of the target node's mypPrev member, like this: pNode ->mypNext->mypPrev = pNode->mypPrev;

[3] See McGregor pages 1-10

[4] See McGregor page 8, lines 8-10

> **Second,** using extended pointer-to-member notation to access the previous node's mypNext member, assign to it the address of the target nodes mypNext member, like this: pNode ->mypPrev->mypNext = pNode->mypNext;"[5]

As can be seen in the cited section, the deletion operation disclosed by McGregor is a two-step-sequential operation. Because even the sub-operations that might be used when updating a node are multi-step operations, McGregor cannot possibly disclose performing an atomic operation for updating a node in a linked list.

Hence, McGregor nowhere discloses performing an atomic operation for lock-free updating an existing node in a linked list. Thus, McGregor cannot render the claimed invention obvious, as described below.

Rejections of Independent Claims 1, 15 and 29

Applicant respectfully notes that Examiner has failed to establish prima facie obviousness because Examiner has failed to explain fundamental differences between the cited McGregor prior art and independent claims 1, 15, and 29 in the instant application. More specifically, Examiner has failed to explain how McGregor's disclosure of a **two-step node-deletion operation** renders obvious the present invention's single **atomic operation** for updating a node in the linked list.

---

[5] See McGregor page 10, lines 1-13

McGregor Nowhere Discloses Performing an Atomic Operation for Updating a Node in a Linked List

The claimed invention performs **a single atomic operation on a linked list to update data in a given node.** The atomic operation includes two sub-operations: (1) inserting a new node into the linked list; and (2) marking an existing node to indicate removal.[6] The atomic operation enables these operations to outwardly appear as a single operation, thereby preventing erroneous accesses of the data within the node.

In the 0903 OA, Examiner averred that McGregor discloses the single atomic operation by reciting "*all you have to do is swap the pointers of the nodes on either side of the node you are deleting*."[7] Applicant respectfully disagrees with the rejection. Applicant respectfully points out that McGregor is disclosing only the simple delete operation for removing a node from the linked list. Moreover, even in describing the delete operation, McGregor nowhere discloses at least two of the features of the atomic operation used in the claimed invention, as detailed below.

1. McGregor Nowhere Discloses a Single Atomic Operation

As indicated above, McGregor nowhere describes a node update in a linked list that involves an atomic operation. Even McGregor's basic "delete" and "insert" operations are not performed atomically. In fact, McGregor expressly describes on page 10 that the pointer swapping operation for deleting a node is a **two-step operation** which includes **two separate sequential steps**. Specifically, McGregor recites:

---

[6] see instant application, paragraph [0033]

[7] see 0903 OA, page 3, paragraph 4

> "All you have to do is swap the pointers of the nodes on either side of the node you are deleting. You swap the pointers in **two steps**.
>
> **First**, using extended pointer-to-member notation to access the next node's mypPrev member, assign to it the address of the target node's mypPrev member, like this: pNode ->mypNext->mypPrev = pNode->mypPrev;
>
> **Second,** using extended pointer-to-member notation to access the previous node's mypNext member, assign to it the address of the target nodes mypNext member, like this: pNode ->mypPrev->mypNext = pNode->mypNext;"[8]

As is well known in computer science, an **atomic operation** refers to a set of operations that can be combined so that they appear to the rest of the system to be a single operation. Thus, the two-step sequential operation in McGregor is not and cannot be an atomic operation. Applicant respectfully points out that in disclosing a **two-step operation, McGregor teaches directly away from the single atomic operation in the claimed invention**.

2. McGregor Describes a Node-Deletion Operation Fundamentally Different than the Update Operation of the Instant Application

McGregor describes an operation fundamentally different from the single atomic operation in the claimed embodiments. Recall that the single atomic operation is used to update an existing node by both: (1) *inserting* a new node into the linked list by modifying the next pointer of the existing node to point to the new node; and (2) *marking* the existing (outdated) node to indicate removal.[9] In contrast, McGregor is limited to *deleting* an existing node from the list by *redirecting* the pointers of the nodes on either side of the existing node.[10]

---

[8] see McGregor, page 10, lines 1-13

[9] see instant application, paragraph [0033]

[10] see McGregor, page 10, lines 1-4

Applicant respectfully points out that there are at least two differences between the claimed embodiments and McGregor. First, the two-step operation in McGregor does not insert a new node into a linked list, whereas the single-atomic operation in the instant application inserts a new node into a linked list. Second, the two-step operation in McGregor removes a node from the linked list, whereas the single-atomic operation in the instant application marks an outdated version node to indicate impending deletion (thereby preventing any of the data in the node from being read in the event that the node is accessed). In fact, in the claimed embodiments, the marked node is not actually completely removed from the linked list until a subsequent splicing operation is performed on the marked node.[11]

Hence, McGregor not only fails to disclose the individual features of the single atomic operation, but fails to disclose combining these features into a single atomic operation.

We now turn to Guthrie, which describes a snapshot system configured to create and make available multiple snapshots representing different states of the data at various times. More specifically, Guthrie describes various techniques for making snapshots of existing nodes by copying the contents of the existing nodes into new nodes and then modifying pointers to add the new nodes into the linked list.[12] However, the snapshots copying process described by Guthrie does not suggest or imply **performing a single atomic operation to achieve the two described objectives:** (1) *inserting* a new node into the linked list by modifying the next pointer of the existing node to point to the new node; and (2) *marking* the existing (outdated) node to indicate deletion.

---

[11] see instant application, paragraph [0036]

[12] see Guthrie. Col. 7, lines 15-20

In summary, the combined system of McGregor and Guthrie nowhere discloses performing a single atomic operation to achieve the two above-described objectives. Hence, the rejection of the independent claims 1, 15, and 29 under 35 U.S.C. § 103(a) based on McGregor in view of Guthrie is incorrect because the combined system is fundamentally distinct from the claimed invention. Applicant, therefore, respectfully requests the withdrawal of the rejection of these claims under 35 U.S.C. § 103(a).

Conclusion

In summary, the combined system of McGregor and Guthrie nowhere discloses performing a single atomic operation to achieve the two above-described objectives: (1) inserting a new node into the linked list by modifying the next pointer of the existing node to point to the new node; and (2) marking the existing (outdated) node to indicate deletion. Hence, the rejection of the independent claims 1, 15, and 29 under 35 U.S.C. § 103(a) based on McGregor in view of Guthrie is incorrect because the Examiner has not sufficiently established the prima facie case of obviousness.

In view of the foregoing, Appellant respectfully requests the reversal of all of the rejections in the Final Office Action mailed 08 April 2008. Appellant further requests allowance of dependent claims 1, 3-14, 16-28, and 30-42.

Respectfully submitted,

By: __/Anthony P. Jones/___
Anthony P. Jones
Registration No. 59,521

Date: 09 February 2009

Anthony Jones
Park, Vaughan & Fleming LLP
2820 Fifth Street
Davis, CA 95618-7759
Tel: (530) 759-1666
Fax: (530) 759-1665
Email: tony@parklegal.com

APPENDICES

Appendix A: Claims Appendix

1. (Previously presented) A method for performing a lock-free update to one or more fields in an existing node in a linked list, comprising:

receiving a reference to the existing node in the linked list, wherein the existing node contains the one or more fields to be updated;

obtaining a new node to be added to the linked list, wherein other processes do not possess references to the new node and therefore cannot initially access the new node;

copying a snapshot of the existing node to the new node, which includes copying a next pointer of the existing node to the new node, so that the new node points to a node immediately following the existing node;

updating one or more fields in the new node that correspond to the one or more fields in the existing node;

performing a single atomic operation that modifies the next pointer of the existing node to point to the new node and also marks the next pointer to indicate that the existing node is deleted, whereby the new node becomes part of the linked list and the existing node is deleted in a single atomic operation; and

splicing the existing node out of the linked list by atomically modifying the next pointer of a node immediately preceding the existing node in the linked list to point to the new node, instead of pointing to the existing node,

wherein copying the snapshot of the existing node to the new node further involves examining the next pointer of the existing node to determine if the existing node has been deleted.

2. (Cancelled)

3. (Previously presented) The method of claim 1, wherein if a process that deleted the existing node does not perform the splicing operation, another process, which subsequently detects that the existing node has been deleted, performs the splicing operation.

4. (Previously presented) The method of claim 1, wherein copying a snapshot of the existing node to the new node involves:

copying the contents of the existing node to the new node;

examining the next pointer of the existing node to determine if the existing node has been deleted; and

if so, taking a remedial action;

otherwise, not taking the remedial action.

5. (Previously presented) The method of claim 4, wherein taking the remedial action involves:

following the next pointer of the existing node in an attempt to find an updated version of the existing node;

if an updated version of the existing node is found, copying a snapshot of the updated version of the existing node to the new node; and

if an updated version of the existing node is not found, indicating that the remedial action fails.

6. (Original) The method of claim 1, further comprising deleting a target node from the linked list by:

receiving a reference to the target node to be deleted from the linked list;

atomically marking a next pointer in the target node to indicate that the target node is deleted; and

atomically modifying the next pointer of a node immediately preceding the target node in the linked list to point to a node immediately following the target node in the linked list, instead of pointing to the target node, thereby splicing the target node out of the linked list.

7. (Original) The method of claim 6, wherein after the target node is spliced out of the linked list, the method further comprises modifying the next pointer of the target node so that the next pointer remains marked but points to a node immediately preceding the target node instead of the node immediately following node the target node in the linked list.

8. (Original) The method of claim 1, further comprising inserting an additional node into the linked list by:

identifying a node immediately preceding the additional node in the linked list;

identifying a node immediately following the additional node in the linked list; and

splicing the additional node into the linked list by,

setting the next pointer for the additional node to point to the immediately following node, and

atomically updating the next pointer of the immediately preceding node to point to the additional node.

9. (Previously presented) The method of claim 1, further comprising reading a snapshot of multiple fields from a target node in the linked list by:

reading the multiple fields from the target node;

examining the next pointer of the target node to determine if the target node has been deleted; and

if so, taking a remedial action;

otherwise, not taking the remedial action.

10. (Previously presented) The method of claim 9, wherein taking the remedial action involves:

following the next pointer of the target node in an attempt to find an updated version of the target node;

if an updated version of the target node is found, repeating the process of reading a snapshot of the multiple fields from the updated version of the target node; and

if an updated version of the existing node is not found, indicating that the remedial action fails.

11. (Original) The method of claim 1, wherein atomically modifying the next pointer of the existing node to indicate that the existing node is deleted involves setting a "deleted bit" in the next pointer.

12. (Previously presented) The method of claim 1, wherein while atomically modifying the next pointer of the existing node,

if the next pointer indicates that the existing node is already deleted, the atomic modification operation fails and the method further comprises taking a remedial action to deal with the fact that the existing node is already deleted;

otherwise, continuing performing the atomic modification operation.

13. (Original) The method of claim 1, wherein a given node in the linked list includes:

a key that contains an identifier for the given node;

one or more fields containing data values or pointers to data values associated with the given node; and

a next pointer that contains the address of a node that immediately follows the given node in the linked list, and that also contains a deleted indicator, which indicates whether the given node has been deleted.

14. (Original) The method of claim 1, further comprising periodically performing a garbage-collection operation to reclaim deleted nodes that have become unreachable.

15. (Previously presented) A computer-readable storage medium storing instructions that when executed by a computer cause the computer to perform a method for performing a lock-free update to one or more fields in an existing node in a linked list, the method comprising:

receiving a reference to the existing node in the linked list, wherein the existing node contains the one or more fields to be updated;

obtaining a new node to be added to the linked list, wherein other processes do not possess references to the new node and therefore cannot initially access the new node;

copying a snapshot of the existing node to the new node, which includes copying a next pointer of the existing node to the new node so that the new node points to a node immediately following the existing node;

updating one or more fields in the new node that correspond to the one or more fields in the existing node;

performing a single atomic operation that modifies a next pointer of the existing node to point to the new node and also marks the next pointer to indicate that the existing node is deleted, whereby the new node becomes part of the linked list and the existing node is deleted in a single atomic operation; and

splicing the existing node out of the linked list by atomically modifying the next pointer of a node immediately preceding the existing node in the linked list to point to the new node, instead of pointing to the existing node,

wherein copying the snapshot of the existing node to the new node further involves examining the next pointer of the existing node to determine if the existing node has been deleted.

16. (Cancelled)

17. (Previously presented) The computer-readable storage medium of claim 15, wherein if a process that deleted the existing node does not perform the splicing operation, another process, which subsequently detects that the existing node has been deleted, performs the splicing operation.

18. (Previously presented) The computer-readable storage medium of claim 15, wherein copying the snapshot of the existing node to the new node involves:

copying the contents of the existing node to the new node;

examining the next pointer of the existing node to determine if the existing node has been deleted; and

if so, taking a remedial action;

otherwise, not taking the remedial action.

19. (Previously presented) The computer-readable storage medium of claim 18, wherein taking the remedial action involves:

following the next pointer of the existing node in an attempt to find an updated version of the existing node;

if an updated version of the existing node is found, copying a snapshot of the updated version of the existing node to the new node; and

if an updated version of the existing node is not found, indicating that the remedial action fails.

20. (Original) The computer-readable storage medium of claim 15, wherein the method further comprises deleting a target node from the linked list by:

receiving a reference to the target node to be deleted from the linked list;

atomically marking a next pointer in the target node to indicate that the target node is deleted; and

atomically modifying the next pointer of a node immediately preceding the target node in the linked list to point to a node immediately following the target node in the linked list, instead of pointing to the target node, thereby splicing the target node out of the linked list.

21. (Original) The computer-readable storage medium of claim 20, wherein after the target node is spliced out of the linked list, the method further comprises modifying the next pointer of the target node so that the next pointer remains marked but points to a node immediately preceding the target node instead of the node immediately following node the target node in the linked list.

22. (Original) The computer-readable storage medium of claim 15, wherein the method further comprises inserting an additional node into the linked list by:

identifying a node immediately preceding the additional node in the linked list;

identifying a node immediately following the additional node in the linked list; and

splicing the additional node into the linked list by,

setting the next pointer for the additional node to point to the immediately following node, and

atomically updating the next pointer of the immediately preceding node to point to the additional node.

23. (Previously presented) The computer-readable storage medium of claim 15, wherein the method further comprises reading a snapshot of multiple fields from a target node in the linked list by:

reading the multiple fields from the target node;

examining the next pointer of the target node to determine if the target node has been deleted; and

if so, taking a remedial action;

otherwise, not taking the remedial action.

24. (Previously presented) The computer-readable storage medium of claim 23, wherein taking the remedial action involves:

following the next pointer of the target node in an attempt to find an updated version of the target node;

if an updated version of the target node is found, repeating the process of reading a snapshot of the multiple fields from the updated version of the target node; and

if an updated version of the existing node is not found, indicating that the remedial action fails.

25. (Original) The computer-readable storage medium of claim 15, wherein atomically modifying the next pointer of the existing node to indicate that the existing node is deleted involves setting a "deleted bit" in the next pointer.

26. (Previously presented) The computer-readable storage medium of claim 15, wherein while atomically modifying the next pointer of the existing node,

if the next pointer indicates that the existing node is already deleted, the atomic modification operation fails and the method further comprises taking a remedial action to deal with the fact that the existing node is already deleted;

otherwise, continuing performing the atomic modification operation.

27. (Original) The computer-readable storage medium of claim 15, wherein a given node in the linked list includes:

a key that contains an identifier for the given node;

one or more fields containing data values or pointers to data values associated with the given node; and

a next pointer that contains the address of a node that immediately follows the given node in the linked list, and that also contains a deleted indicator, which indicates whether the given node has been deleted.

28. (Original) The computer-readable storage medium of claim 15, wherein the method further comprises periodically performing a garbage-collection operation to reclaim deleted nodes that have become unreachable.

29. (Previously presented) An apparatus that performs a lock-free update to one or more fields in an existing node in a linked list, comprising:

a node obtaining mechanism configured to obtain a new node to be added to the linked list, wherein other processes do not possess references to the new node and therefore cannot initially access the new node;

a copying mechanism configured to copy a snapshot of the existing node to the new node, which includes copying a next pointer of the existing node to the new node so that the new node points to a node immediately following the existing node;

an updating mechanism configured to update one or more fields in the new node that correspond to the one or more fields in the existing node;

a modification mechanism configured to perform a single atomic operation that modifies a next pointer of the existing node to point to the new node and also marks the next pointer to indicate that the existing node is deleted, whereby the new node becomes part of the linked list and the existing node is deleted in a single atomic operation; and

a splicing mechanism configured to splice the existing node out of the linked list by atomically modifying the next pointer of a node immediately preceding the existing node in the linked list to point to the new node, instead of pointing to the existing node,

wherein copying the snapshot of the existing node to the new node further involves examining the next pointer of the existing node to determine if the existing node has been deleted.

30. (Cancelled)

31. (Previously presented) The apparatus of claim 29, wherein if a process that deleted the existing node does not activate the splicing mechanism, another process, which subsequently detects that the existing node has been deleted, activates the splicing mechanism.

32. (Previously presented) The apparatus of claim 29, wherein the copying mechanism is configured to:

copy the contents of the existing node to the new node;

examine the next pointer of the existing node to determine if the existing node has been deleted; and

if so, to take a remedial action;

otherwise, not taking the remedial action.

33. (Previously presented) The apparatus of claim 32, wherein while taking the remedial action, the copying mechanism is configured to:

follow the next pointer of the existing node in an attempt to find an updated version of the existing node;

if an updated version of the existing node is found, to copy a snapshot of the updated version of the existing node to the new node; and

if an updated version of the existing node is not found, indicating that the remedial action fails.

34. (Original) The apparatus of claim 29, further comprising a deletion mechanism configured to delete a target node from the linked list, wherein the deletion mechanism is configured to:

receive a reference to the target node to be deleted from the linked list;

atomically mark a next pointer in the target node to indicate that the target node is deleted; and to

atomically modify the next pointer of a node immediately preceding the target node in the linked list to point to a node immediately following the target node in the linked list, instead of pointing to the target node, thereby splicing the target node out of the linked list.

35. (Original) The apparatus of claim 34, wherein after the target node is spliced out of the linked list, the deletion mechanism is configured to modify the next pointer of the target node so that the next pointer remains marked but points to a node immediately preceding the target node instead of the node immediately following node the target node in the linked list.

36. (Original) The apparatus of claim 29, further comprising an insertion mechanism configured to insert an additional node into the linked list, wherein the insertion mechanism is configured to:

identify a node immediately preceding the additional node in the linked list;

identify a node immediately following the additional node in the linked list; and to

splice the additional node into the linked list by,

setting the next pointer for the additional node to point to the immediately following node, and

atomically updating the next pointer of the immediately preceding node to point to the additional node.

37. (Previously presented) The apparatus of claim 29, further comprising a reading mechanism configured to read a snapshot of multiple fields from a target node in the linked list, wherein the reading mechanism is configured to:

read the multiple fields from the target node;

examine the next pointer of the target node to determine if the target node has been deleted; and

if so, to take a remedial action;

otherwise, not taking the remedial action.

38. (Previously presented) The apparatus of claim 37, wherein while taking the remedial action, the reading mechanism is configured to:

follow the next pointer of the target node in an attempt to find an updated version of the target node;

if an updated version of the target node is found, to repeat the process of reading a snapshot of the multiple fields from the updated version of the target node; and

if an updated version of the existing node is not found, indicating that the remedial action fails.

39. (Original) The apparatus of claim 29, wherein while atomically modifying the next pointer of the existing node to indicate that the existing node is deleted, the modification mechanism is configured to set a “deleted bit” in the next pointer.

40. (Previously presented) The apparatus of claim 29, wherein while atomically modifying the next pointer of the existing node,

if the next pointer indicates that the existing node is already deleted, the modification mechanism is configured to:

fail the modification operation fails; and to

take a remedial action to deal with the fact that the existing node is already deleted;

otherwise, continue performing the atomic modification operation.

41. (Original) The apparatus of claim 29, wherein a given node in the linked list includes:

a key that contains an identifier for the given node;

one or more fields containing data values or pointers to data values associated with the given node; and

a next pointer that contains the address of a node that immediately follows the given node in the linked list, and that also contains a deleted indicator, which indicates whether the given node has been deleted.

42. (Original) The apparatus of claim 29, further comprising a garbage collection mechanism configured to periodically perform a garbage-collection operation to reclaim deleted nodes that have become unreachable.

Appendix B: Evidence

For this appeal, Appellants do not rely on any evidence submitted pursuant to §§ 1.130, 1.131, or 1.132, or other evidence entered by the Examiner.

Appendix C: Related Proceedings

Appellants are aware of no related proceedings.